SN Computer Science (2025) 6:480
https://doi.org/10.1007/s42979-025-04015-z

ORIGINAL RESEARCH

CrazyKhoreia: A Robotic Perception System for MAV Path Planning From Digital Images

Santiago Restrepo-Garcia[1] · Victor Romero-Cano[2]

Received: 14 April 2024 / Accepted: 1 May 2025

Abstract

Micro Air Vehicles (MAVs) can be used for a wide range of applications, such as drone-based light shows for cultural or marketing purposes, or as a replacement for fireworks. However, more widespread usage of this technology is precluded by the usability gap between choreography design and deployment on target drones. Seamless deployment of MAV choreographies requires the building of computational interfaces that can obtain drone trajectories from conventional media such as digital images. In this paper, we propose CrazyKhoreia, a low-cost, computationally efficient approach to MAV choreography design. CrazyKhoreia is a robotic perception system that obtains a safe, traversable, and accurate waypoint matrix from a digital image. We validate our trajectory generation system through two distinct modes of operation: light painting, where an MAV flies through all waypoints, and multi-drone formation, in which multiple MAVs are arranged to emulate a given image. The utility of each mode is evaluated differently, using the full resolution CrazyKhoreia output as a reference for comparison. For the light painting mode, we logged the MAV's pose at a frequency of 10 Hz and compared it with the reference for different levels of detail in the reproduced path. The Root Mean Squared Error (RMSE) ranges from 0.1287 to 0.3190 m. For multi-drone formation, where the swarm remains stationary, we computed the RMSE by comparing observed positions with the reference across multiple tests, resulting in a mean RMSE value of 0.1360 m.

Keywords Boundary tracing · Computer vision · Drone swarm · MAV · Path planning · Robotic perception

Introduction

Micro Air Vehicles (MAVs) have been used in numerous applications in daily life and industry, including racing, cinema, photography, precision agriculture, shipping, security, military defense, data gathering, and fire control, among others [1]. In addition to these scenarios, MAVs can also replace fireworks for entertainment or marketing purposes. Many firms have created impressive light shows with thousands of drones, where a remote centralized ground station typically controls a large MAV swarm by transmitting choreography commands to each MAV.

The use of MAVs for light shows offers significant advantages over conventional fireworks. MAVs are reusable, and do not rely on any class of combustion material, meaning that they do not leave particulate matter residues in the same way as fireworks. Furthermore, the interest of many governments around the world in producing more environmentally friendly shows has created a new sector in the market for MAV light shows.

In general, to create a formation for MAV light shows, multiple waypoints must be established manually to guide the MAVs to mimic figures such as business logos, geometric shapes, or objects. However, this light show technique typically requires a substantial number of MAVs and a significant flight area to allow the show to be performed safely and the intended pattern to be accurately portrayed [2].

A closely related application is Light Painting (LP), a photography technique that involves moving a light source (such as a MAV equipped with a light bulb) while a camera sensor is exposed with a slow shutter speed. With this

✉ Victor Romero-Cano
romerocanov@cardiff.ac.uk

Santiago Restrepo-Garcia
santiago.restrepo_g@uao.edu.co

[1] Faculty of Engineering, Universidad Autónoma de Occidente, Cll 25 115-85, Cali 760030, Valle del Cauca, Colombia

[2] School of Computer Science and Informatics, Cardiff University, Cardiff CF24 4 AG, Wales, UK

Published online: 20 May 2025

method, it is possible to recreate the figures mentioned above with a single MAV. In this case, the path planning problem involves defining the trajectory required for the MAV to create a drawing using a light bulb and a stationary slow-shutter camera.

Although the manual design of MAV choreographies may work as intended and produce impressive light shows, it is a prolonged process that requires extensive manual labor and many designers and engineers. The cost of creating a MAV light show increases with the design complexity and the number of MAVs in flight, which creates a usability gap between choreography design and the deployment of MAVs in the field, thereby limiting the more frequent usage of this technology. It is necessary to incorporate intelligent robotic systems into the choreography design process to close this gap and to reduce the cost of using this technology.

The development of robotic perception systems has enabled robots to process and interpret sensory data, allowing them to model their environment and make decisions in the physical world, including path planning. Many of these systems take advantage of machine learning (ML) and computer vision technologies [3]. For example, Saavedra et al. [4, 5] used robotic perception to autonomously land UAVs on designated platforms. When implemented effectively, perception systems can enhance automation, improve security, and enable more efficient MAV choreography design through advanced computational interfaces.

A previous version of this manuscript was posted as a preprint in 2023 [6], introducing a robotic perception system that can extract waypoints from standard 8-bit color or grayscale images. The extracted waypoints can be used in both Multiple Drone Formation (MDF) and LP applications, as shown in Fig. 1.

The contributions of this work are as follows.

- We present a functional, open source, multi-platform robotic perception architecture for MAV path planning from digital images, which is entirely implemented in Python.
- We introduce a documented, easily scalable library for image and contour processing for MAVs.
- We present an end-to-end pipeline that works in two modes of operation. MDF, a collision-free swarm coordination algorithm for dense environments and LP, a cost-conscious approach for aerial light shows.

Related Work

The task of path planning for MAVs from digital images aims to solve two problems: the generation of a trajectory from high-level user input, and collision avoidance between drones.

In [7], the authors proposed a trajectory planner that ensured a smooth, safe transition between established patterns defined by motion primitives (i.e., a wave surface equation, a helix or a cone). Although motion primitives allow the user to set the desired trajectory by means of a mathematical equation, this requires the end user to have a knowledge of surface equations and motion dynamics. In contrast, our work presents an easy-to-use architecture in which the end user does not need to specify motion priors.

The authors of [8] developed a picture-based motion planner for a drawing robot. Their system used Canny's algorithm [9] to detect edges from a digital image. However, this algorithm has limitations in the context of path planning, as it does not generate continuous paths. To address this issue, the authors devised an improved path planning algorithm that scanned and tracked the white pixels in the image, categorizing them as start/end, normal, or branch points. As an

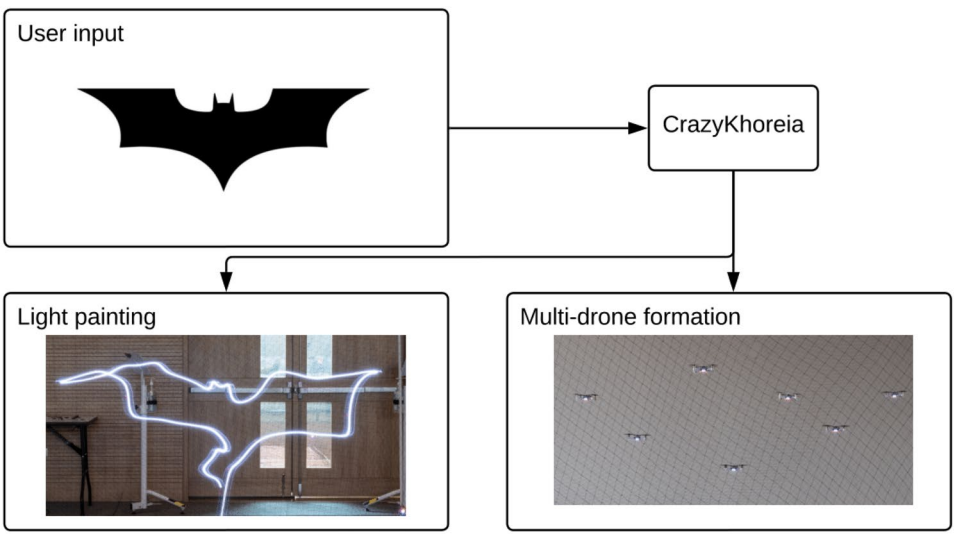

Fig. 1 Examples of the use of CrazyKhoreia. A single digital image (in this example the silhouette of a bat) and the flight space characteristics are passed as input to CrazyKhoreia, which applies several computer vision algorithms to the image to generate a flight plan for light painting or multi-drone use

alternative, the author of [10] proposed a boundary tracing algorithm that naturally overcame this issue without requiring additional computation steps. We therefore employed this boundary-tracing approach in our scheme.

Incorporating an automatic perception and planning tool into Computer Numerical Control (CNC) machines is beneficial, as it enables automatic tool-path generation on raw materials for machining. By vectorizing an image and utilizing tool path generation software such as Mastercam, a G Code program can be created that gives the required trajectories. In a similar vein, the computer art algorithm proposed in [11] utilized a monochrome digital image to generate line and cutout art by imitating its patterns. In this algorithm, the size of the lines and cutouts at each point was determined by the pixel intensity. Likewise, the authors of [12] used a Crazyflie MAV to create stippled prints from centroidal Voronoi diagrams and optimized them with an approximation of the travelling salesman problem to paint the input image. The algorithms used in [11, 12] formed a source of inspiration for the architecture of our work, as we decided to restrict the user input to a digital image, the dimensions of the flight space, and the characteristics of the MAV. However, the algorithms described above only worked with monochromatic images; in contrast, Crazykhoreia can handle both monochromatic and color images, thereby increasing the adaptability and usability of the system.

Several works have focused on trajectory generation given a user input; for instance, in [13], the authors used a Deep Neural Network (DNN) to recognize hand gestures predefined by the user from a camera to control the path of an MAV for LP. In [14], the authors used ML-based processing of IMU data for gesture classification instead. These algorithms are limited to the use of predefined gestures and trajectories, meaning that the end user can select from only a small range of options. Crazykhoreia does not have a limited number of trajectories, which increases the possible outputs of the system.

In [15], the authors described the creation of Airways, an interactive MAV trajectory generator designed for entry-level users without a technical background. Airways can control the trajectory of an MAV based on high-level input from the user, such as drawings, goal positions for video panning, games, LP, and aerial shots. This algorithm creates feasible, collision-free trajectories based on high-level input optimizations using iterative quadratic programming, which solves the problem of MAV control. Obstacles are modeled as spheres and used to adjust the path of the MAV in each iteration. This obstacle-avoidance approach may present issues when multiple MAVs fly in dense formations due to the aerodynamic properties of the aircraft, such as the downwash effect, which is significant beneath the rotors of a quadcopter [16]. This effect may cause sufficient instability to break the entire formation, and it is therefore necessary to use another geometrical representation that takes into consideration the downwash effect.

In [2], the authors represented unsafe flight regions by means of an artificial potential field, which the MAV used to avoid collisions; however, this algorithm often became stuck in local optima, thus failing to converge to a feasible solution. The authors of [17] opted to work with buffered Voronoi cells rather than potential fields, resulting in a computationally low-cost algorithm that only required positional data on each MAV in order to avoid collisions. In the Crazyflie firmware, this algorithm was implemented as **collision_avoidance.c**; however, this firmware does not offer Python development, which would have allowed us to use it in our system. We therefore chose to approximate ellipsoids as cuboids, and employed a three-dimensional intersection over union approach to detect collisions between them.

Recent work related to aerial light shows and swarm coordination shows a desire in the community to build this type of systems. Fagundes et al. [18] developed an LP pipeline to take in digital images or parametric curves and processed the resulted motion using videos. In 2023, the authors of [19] implemented a closely related motion-planner by capturing drawn trajectories with a cursor on a GUI. In [20] the authors used three-dimensional ambigrams to establish multi-view aerial light show using swarms while minimizing the quantity of required drones. Finally, [21] presented a solution to the high-complexity concerns involved in aerial light shows by implementing a fault-tolerant task assignment algorithm.

Crazykhoreia

System Architecture

In this subsection, we describe the architecture of the CrazyKhoreia system, and explain how the internal modules work and interact with each other. CrazyKhoreia has two flight modes, LP and MDF. Despite the differences between them in terms of how the MAVs fly, the initial processing steps of the algorithm are the same. CrazyKhoreia first processes the high-level input from the user, in the form of a standard digital image, to obtain contours or object boundaries, and these are then scaled according to the flight space constraints, which are also input by the user. The waypoints are then extracted from the prepared contours.

Both the LP and MDF flight modes rely on these generated waypoints, although the handling differs in each case. In LP, the system relies on the waypoints as input, as they serve as key reference points for the movement of the MAV along the desired path.

To ensure precise and efficient path planning, we consider a user-defined parameter called the "detail", which is determined based on the Euclidean distance between a pair of waypoints. By defining a level of detail based on this distance, the system establishes criteria for limiting the number of waypoints, which affects the level of detail and the smoothness of the MAV's trajectory.

By carefully adjusting the density of the waypoints to achieve the desired resolution, the LP system reaches an optimal balance between accuracy and computational efficiency, according to the precision constraints of the physical system. This approach allows the MAV to transverse the LP trajectory while minimizing the computational cost.

When used in the MDF mode, the algorithm estimates the following quantities:

- An initial grid, based on the number of MAVs and the flight space dimensions.
- An ideal formation, which is created by grouping the waypoints into as many clusters as the number of MAVs available.
- An adjusted MAV formation, which is calculated from the ideal by taking into account unsafe flight areas (in the form of cuboids) to ensure a collision-free formation.
- The assignment of the MAVs to their final positions.

In order to ensure that two or more MAVs do not fly too close to each other, we define a three-dimensional cuboid around each MAV that represents the Unsafe Flight Area (UFA). This region is described using three parameters, which are used to build a three-dimensional cuboid with shape $[length_X, width_Y, height_Z]$ meters from the MAV's center of mass, with the Z axis pointing downwards, to take into consideration the downwash effect produced by the MAV's rotors. Through the use of the UFA, we can adjust the positions of the MDF to assure collision-free swarm flight without warping the resulting formation.

The modules of the system are illustrated in the logical architecture in Fig. 2.

Our system was tested with Crazyflies MAVs, which were connected wirelessly using 2.4 GHz radio communication via a Crazyradio PA 2.4 GHz antenna to a Ubuntu 22.04 PC using the USB protocol, as shown in the physical architecture in Fig. 3.

To make the user's experience smoother, we also designed a GUI to handle the connection between CrazyKhoreia and the Crazyswarm ROS server, which can be difficult for non-technical users. As shown in Fig. 4, this interface uses the waypoint file generated previously. Depending on the flight mode, it automatically writes the necessary data to the Crazyswarm configuration files and executes a flight script, in conjunction with a final user confirmation and a simulation dial that toggles between a simulation and the physical robots; this launches the ROS server and lets the MAV swarm fly.

The proposed scheme is illustrated in Fig. 5, in the form of a Python-based Object Oriented Programming (OOP) software architecture. Each software processing step is

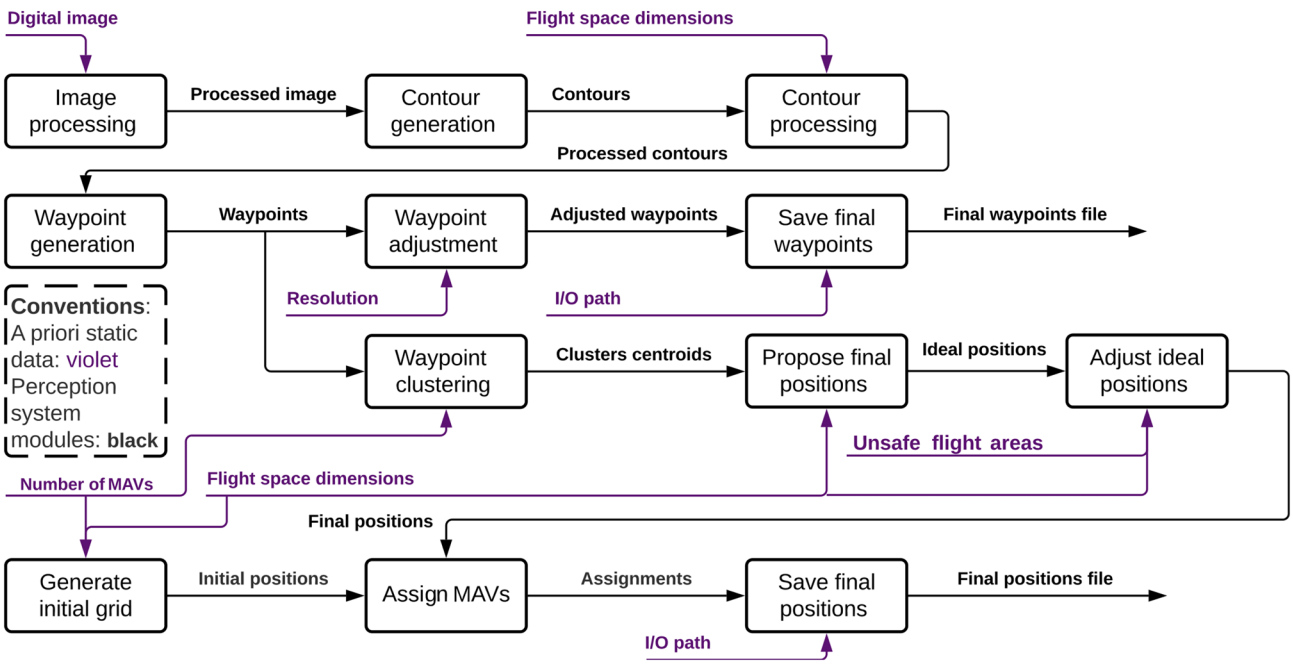

Fig. 2 Logical architecture of CrazyKhoreia. Text and arrows in violet represent the user input, and internal processing is shown in black. This architecture requires minimal user intervention, as the data are only needed once. As output, the user receives a file containing the flight plan

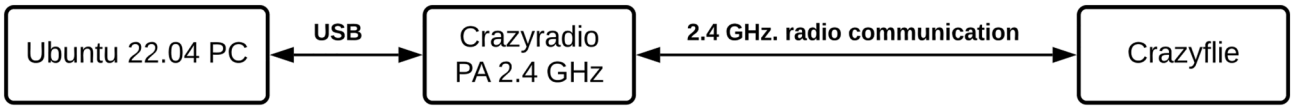

Fig. 3 Physical architecture of CrazyKhoreia. All physical connections are two-way, allowing for constant data exchange between all parties. The computer interfaces with the antenna via USB, and the antenna communicates with each MAV via radio at 2.4 GHz

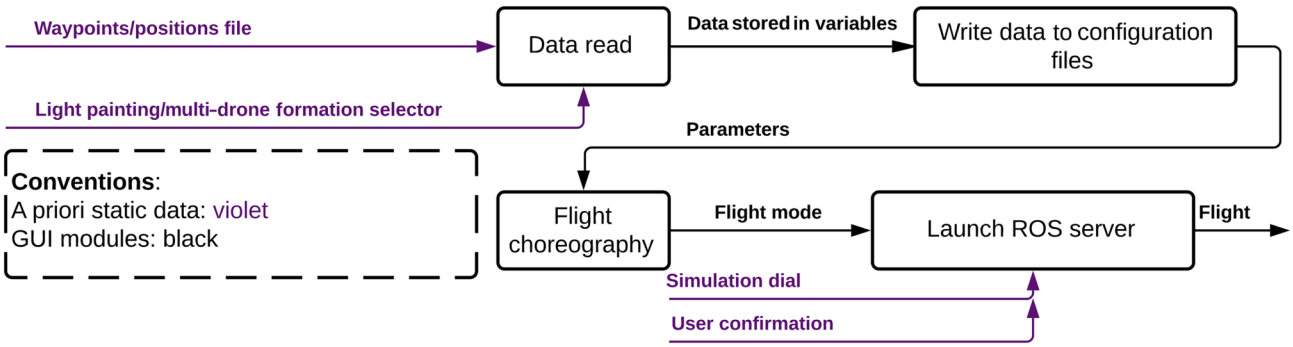

Fig. 4 Logical architecture of CrazyGUI. The GUI connects CrazyKhoreia to the Crazyflies systems from the flight plan file, and the user then chooses to whether to run a simulation or a physical flight

implemented in a different OOP class; for instance, the image-based waypoint generation stage is implemented in the **crazyKhoreia** class, while the LP and MDF functionalities are implemented in the **lightPainting** and **multiDroneFormation** classes, respectively.

Image-Based Waypoint Generation

In our system, we use Suzuki's boundary tracing algorithm [10] to obtain coordinate chains from image pixels, and implement this using the **findContours()** method from OpenCV. This algorithm finds and retrieves contours from a binary 8- bit channel image; however, the input image must be binarized, meaning that the pixel values can only be zero or one

In order to allow CrazyKhoreia to accept any standard digital image, we use Otsu's binarization method [22] to generate a binary image from grayscale or color images before **findContours()** is executed. This method automatically chooses an optimal threshold for the binarization of a monochromatic image by maximizing the variance between the zero and one pixel classes. This is also implemented by OpenCV in the **threshold** module, for which the **THRESH_OTSU** flag should be enabled.

The output of our computer vision pipeline is illustrated in Fig. 6. The original colored image is thresholded using Otsu's method, and the contours are then generated from the correlation between the zero and one pixel classes. These contours are highlighted in green.

A close-up of these contours is presented in Fig. 7, which shows the unprocessed contours.

We note that the frame of the image was recognized as a contour, which is not a desired output, meaning that further processing is required. In this process, as the units of the contours are the width and height of the image, we adjust the pixel-to-distance proportions by using the flight dimensions as constraints and the image width and height to calculate the ratio shown in Eq. (1) and hence scale the contours without affecting the original width-height ratio. The outer frame is deleted, as shown in Fig. 8.

$$P_{x,y} = \frac{Image\ Shape_{x,y}}{Max\ Flight\ Space_{x,y} - Min\ Flight\ Space_{x,y}} \quad (1)$$

In the following sub-sections, we explain the implementation of LP and MDF with the processed contours.

Light Painting

In LP, one MAV flies following the generated waypoints. Since **findContours** generates too many contours, they cannot be immediately used as waypoints, and it is necessary to filter out some of them to save memory space and avoid saturating the control system. Algorithm 1 shows the optimization process, which consists of calculating the Euclidian distance between a pair of waypoints i and $i+1$, and comparing it against the detail factor, as mentioned in Sect. "System Architecture". If the Euclidian distance is less

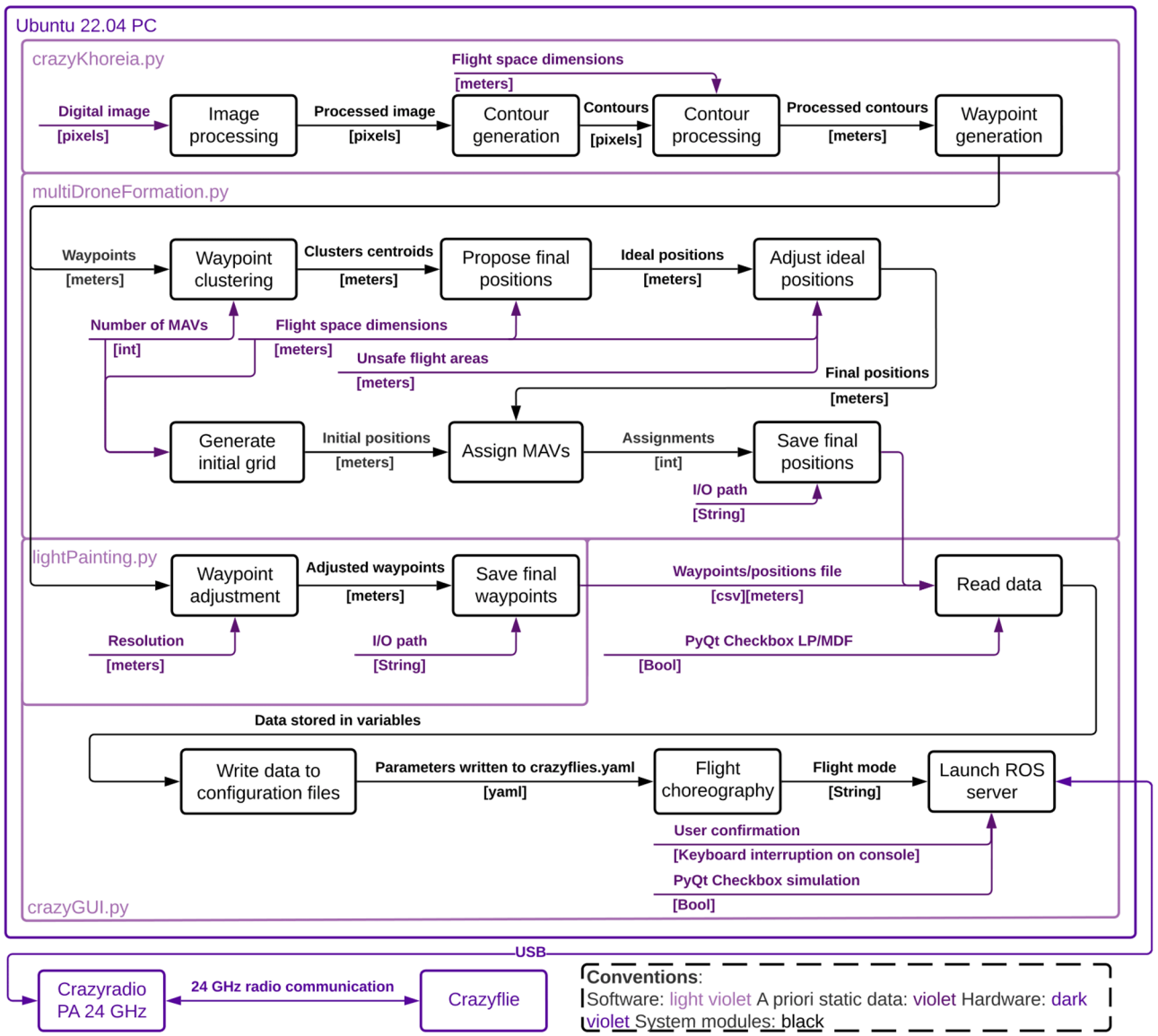

Fig. 5 Logical architecture of CrazyKhoreia with mapping onto the physical architecture. The complete architecture is shown, with the pipeline for a digital image and parameters for the final flight. There are three main modules: CrazyKhoreia (for image-based waypoint generation), MultiDroneFormation (for estimating the end positions to mimic the original image) and LightPainting (for adjusting the waypoints according to a user-defined resolution)

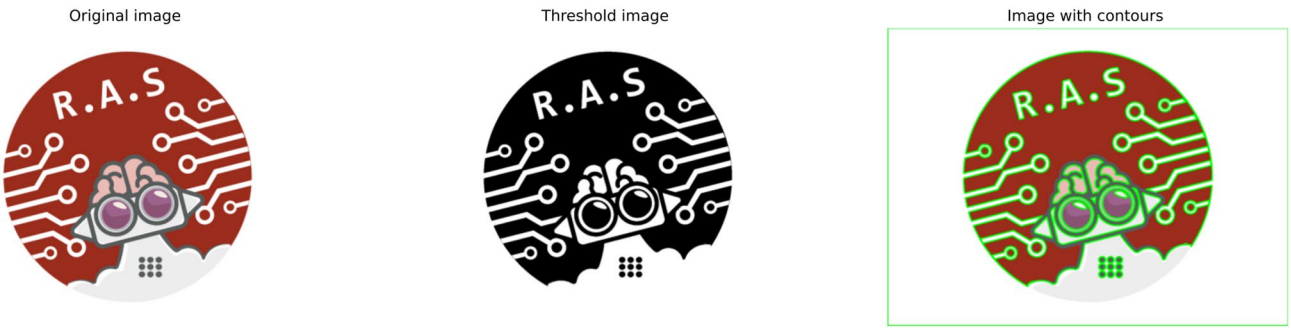

Fig. 6 Image-based waypoint generation. An original image is fed into a pair of computer vision algorithms to obtain a threshold image and a set of unprocessed contours

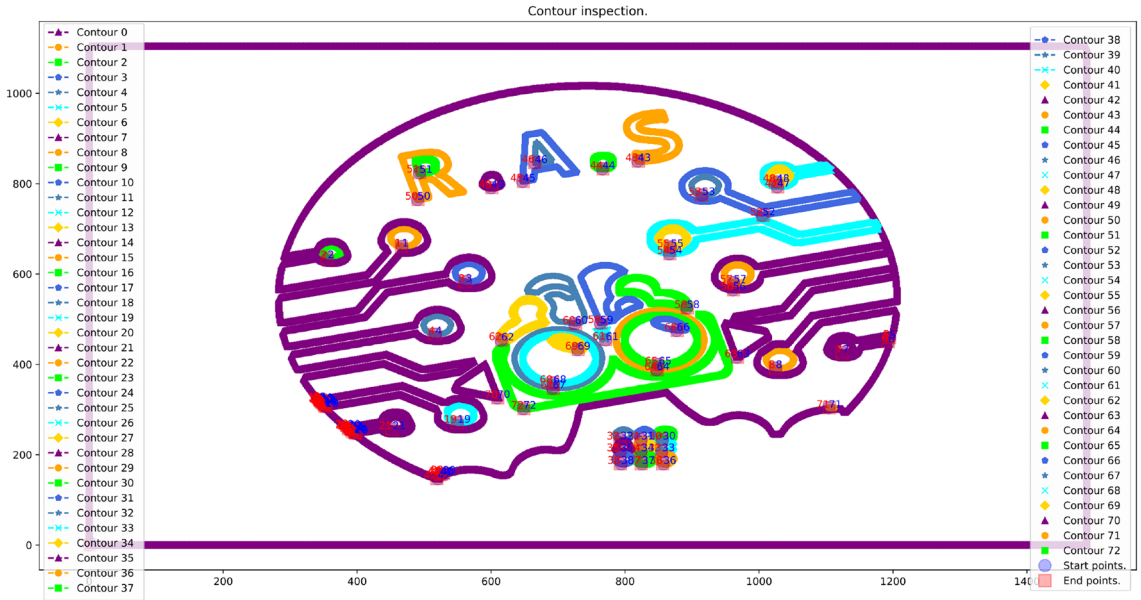

Fig. 7 Raw contours. In this image, there are 72 contours, represented by combinations of colors and shapes as listed on both sides of the graph. Some are unintended contours representing the frame of the image. The coordinate units are pixels

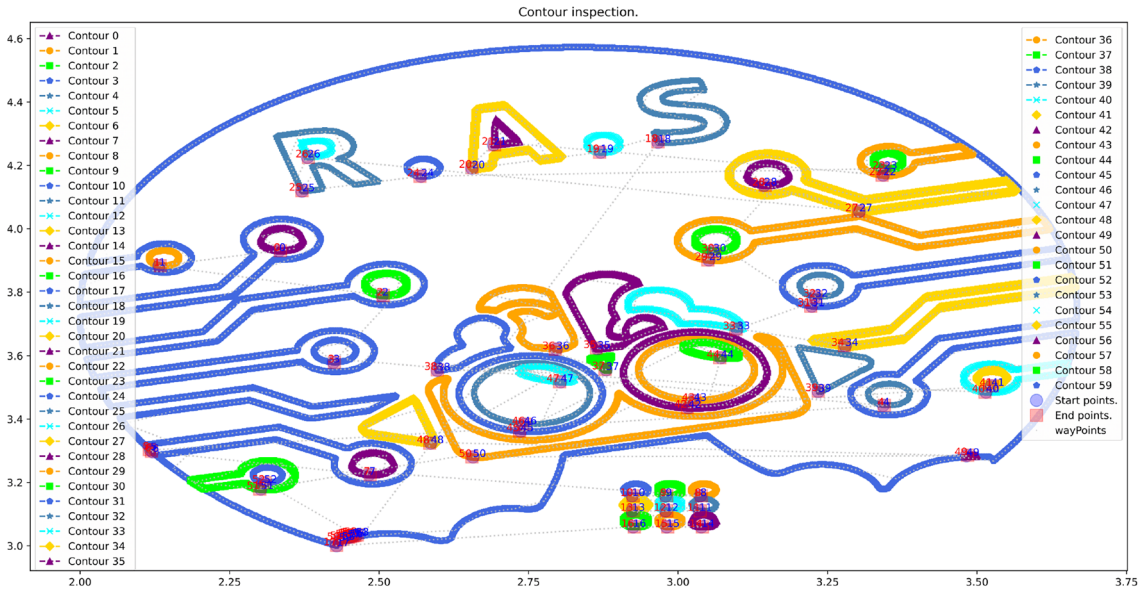

Fig. 8 Raw contours. After contour processing, the contours are scaled in meters, and the extra frame contours are discarded. The fine gray dashed lines represent the MAV's flight plan. There are waypoints in this flight plan that were not originally intended, which happens each time the algorithm changes from one contour to another; however, an optimization strategy for minimizing this effect falls outside the scope of this paper

than the detail threshold, the waypoint i is removed from the waypoint array. We use the image in Fig. 6 to illustrate the effects of this algorithm by setting the level of detail to 0, 0.02, and 0.1 m.

Algorithm 1 LP optimization. A waypoint is removed if the defined distance is below an arbitrary threshold from the next waypoint

```
for waypoint in waypoints do
    if waypoint_i − waypoint_{i+1} < detail then
        delete waypoint_i
    end if
end for
```

The example in Fig. 6 is shown for reference in Fig. 9a, with a detail level of 0 m, and in Fig. 9b, with a value of 0.02 m. In Fig. 9c, the level of detail is set to the average error of the Loco positioning system, i.e. 0.1 m. It can be seen that an increase in the detail variable reduces the overall quality of the expected results; it is important to note that the system error will depend on the capabilities of the aircraft and the flight precision.

Finally, the flight choreography was tested in real life using the Crazyflies MAV described above and shown in Fig. 10. The reader may notice the resemblance between the expected results at a detail level of 0.1 m, shown in Fig. 9(c), and the physical outcome.

Multi-Drone Formation

In the MDF mode, several MAVs are arranged into positions to reconstruct the user's desired figure. In this case, the figure is a digital image provided by the user. The system achieves this in four main steps after image processing: initial grid estimation, clustering, collision-free waypoint adjustment, and MAV assignment.

In this section, we explain these main four steps, and demonstrate them using a bat silhouette image processed to obtain waypoints, as shown in Fig. 11. We conduct physical experiments using the generated waypoints with seven MAVs.

Initial Grid Estimation

In this step, a squared grid is automatically calculated to allow the MAVs to be easily placed in the take-off zone before a flight. This algorithm requires the dimensions of the flight space, the number of MAVs, and the UFA in order to build a centered grid. It first calculates the centre of the flight space, and then estimates the grid size by rounding the squared root of the number of MAVs. Finally, it uses the UFA to calculate the grid initial vertex to which the MAVs are appended in sequential order. A graphical representation of this feature is shown in Fig. 14.

Clustering

To generate the ideal drone positions, the MDF mode uses a well-known clustering algorithm to group the waypoints obtained from the vision algorithms in Sect. 3.2. We use a K-means implementation from the open-source scikit-learn Python library in [23] as a clus- tering strategy to group the previously generated waypoints into N clusters, where N is the number of agents in flight. The system uses the cluster centroids as the ideal MDF positions, and arranges them into an N x 3 matrix. Equation 2 shows the K-means formula; this algorithm is applied to seven MAVs, as shown in Fig. 12.

$$\sum_{i=0}^{N} \min_{\mu_j \in C}(||x_i - \mu_j||^2) \tag{2}$$

Collision-Free Waypoint Adjustment

Since the ideal MDF positions may not allow for a collision-free flight path, they need to be adjusted to avoid collisions. A C++ implementation of the algorithm in [17] uses ellipsoids to describe unsafe flight regions. In this study, we build a virtual cuboid for each MAV in the system, the dimensions of which are defined by the UFA, which represents the minimum safety distance along each axis needed for one MAV to pass another safely.

After building the cuboids for all MAVs, the algorithm uses the Intersection over Union (IoU), also known as the Jaccard index, as shown in Eq. (3). This returns a number between zero and one, according to the volume of overlap between two cuboids A and B, where zero indicates no overlap, and one represents full overlap:

$$IoU = \frac{|A \cap B|}{|A \cup B|} = \frac{|I|}{|U|} \tag{3}$$

Using the IoU metric, and assuming the X-axis is the principal axis, we can adjust the flight path by assuming a fixed

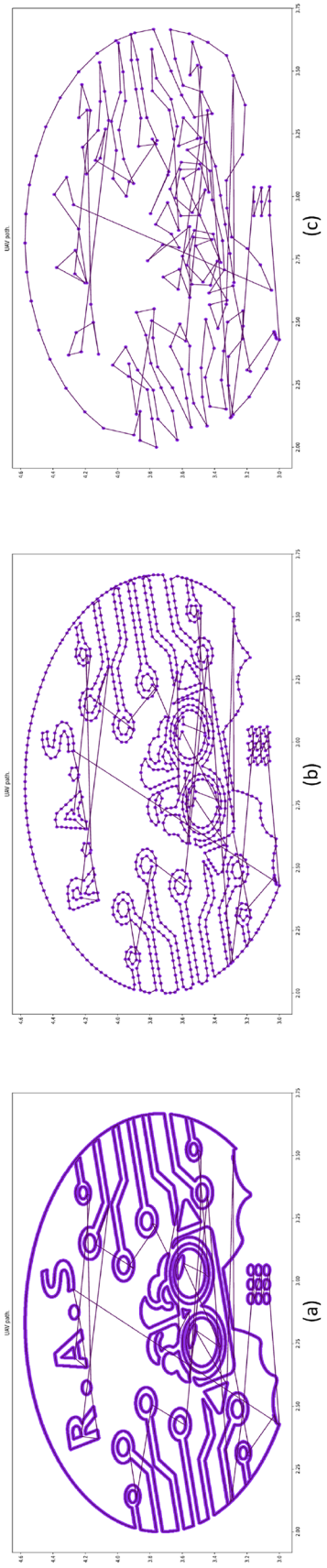

Fig. 9 Comparative of LP mode set with different detail levels. **a** LP with a detail level of zero. At this level, there is no waypoint removal, and it appears as if the physical MAV would fly with almost no error. **b** LP with detail level 0.02. There is a subtle difference from the plan with a detail level of zero, as the distance between waypoints is larger. **c** LP with a detail level of 0.1. The image is no longer recognizable, and the apparent error is significantly greater. This level of detail should be similar to the accuracy of the navigation system

point of view on the YZ plane, so the positions can be translated across the X-axis without significantly compromising the quality of the swarm formation. We can then formulate the path-planning cost function in Eq. (6) as follows.

Let $\mathbf{\Lambda}$ be a list of N λ_i drones. $\begin{pmatrix} \Lambda \\ k \end{pmatrix}$ is a $k \times 2$ array containing all possible drone combinations. Finally, our cost function $\mathbf{J}$ corresponds to the IoU between pairs of MAVs with UFA $\mathbf{\Omega}$.

$$\mathbf{\Lambda} = [\lambda_0, ..., \lambda_N] \tag{4}$$

$$min\ J = \frac{\left| \Omega\begin{pmatrix} \Lambda \\ k \end{pmatrix}_k \cap \Omega\begin{pmatrix} \Lambda \\ k \end{pmatrix}_{k+1} \right|}{\left| \Omega\begin{pmatrix} \Lambda \\ k \end{pmatrix}_k \cup \Omega\begin{pmatrix} \Lambda \\ k \end{pmatrix}_{k+1} \right|} \tag{5}$$

$$\mathbf{J} = [J_0, ..., J_k] \tag{6}$$

When the magnitude of the cost function vector is a minimum, this means that the MAV formation is collision-free and can be used for flight; otherwise, there will be a set of cuboids $\mathbf{\Omega}$ whose IoU is not zero, which represents a collision risk.

The virtual results of this iterative algorithm when applied to the input image in Fig. 11 are illustrated in Fig. 13, where the blue points represent the ideal formation positions obtained from clustering (Fig. 12), and the orange points show how the optimization algorithm has moved the blue dots across the X-axis.

MAV Assignment

In the MDF mode, the MAV assignment module solves the problem of designating a unique MAV to each final MDF position from the initial grid. This assignment requires an intersection-free path that minimizes the overall flight distance based on the cost function. In [24], the authors proposed an optimal allocation approach based on the Hungarian method; however, rather than adjusting the formation of MAVs to create a collision-free and aerodynamically safe configuration, they first checked whether the Euclidian distance between the ideal positions satisfied a predetermined minimum, and manual processing was required to adjust the minimum distance hyperparameter. In our scheme, this process is done automatically using a collision-free waypoint adjustment.

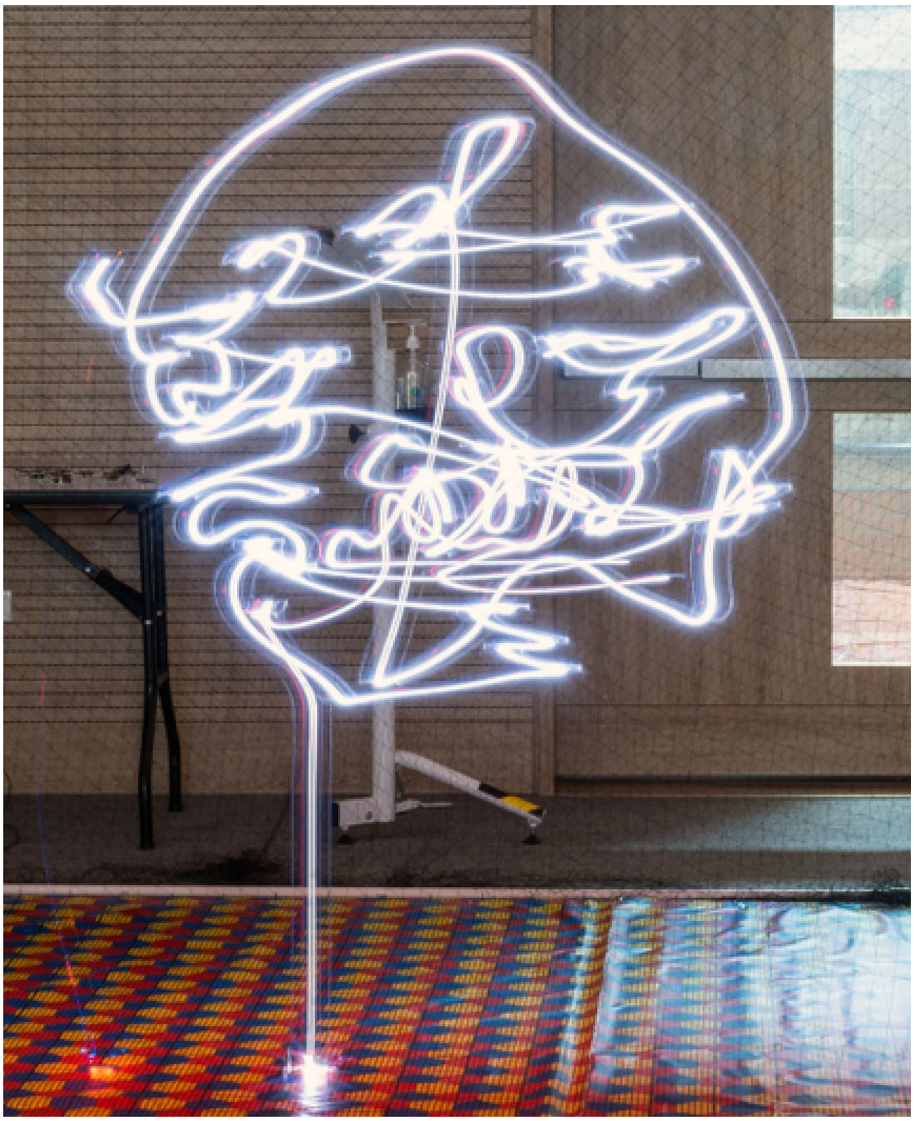

Fig. 10 Real-life results from the LP system. As the physical drones and perception system used to test our system had an accuracy of around 0.1 m, it can be seen that the results of this trial resemble Fig. 11, as the level of detail is similar to the physical accuracy

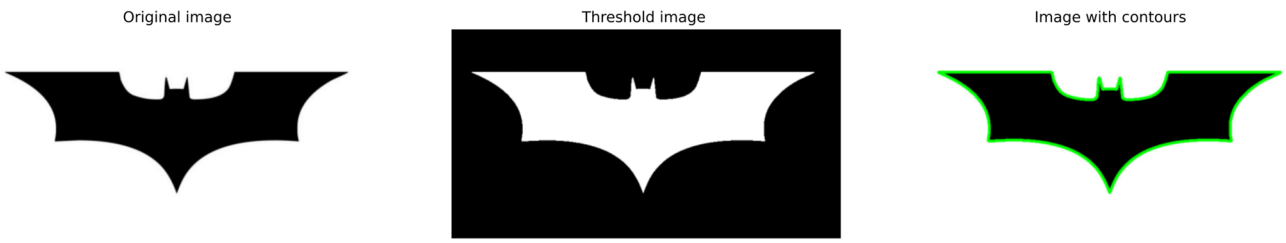

Fig. 11 Image-based waypoint generation. Image used to test the Multi-Drone Formation functionality and its contour processing

The MAV assignment algorithm used in this paper was an implementation of the Hungarian method in the scikit-learn library **linear_sum_assignment**(), which worked as expected in terms of allocating MAVs to their final positions. Figure 14 shows the overall flight plan with the initial grid, flight space dimensions, center of the flight space, ideal and adjusted positions, and a flight path.

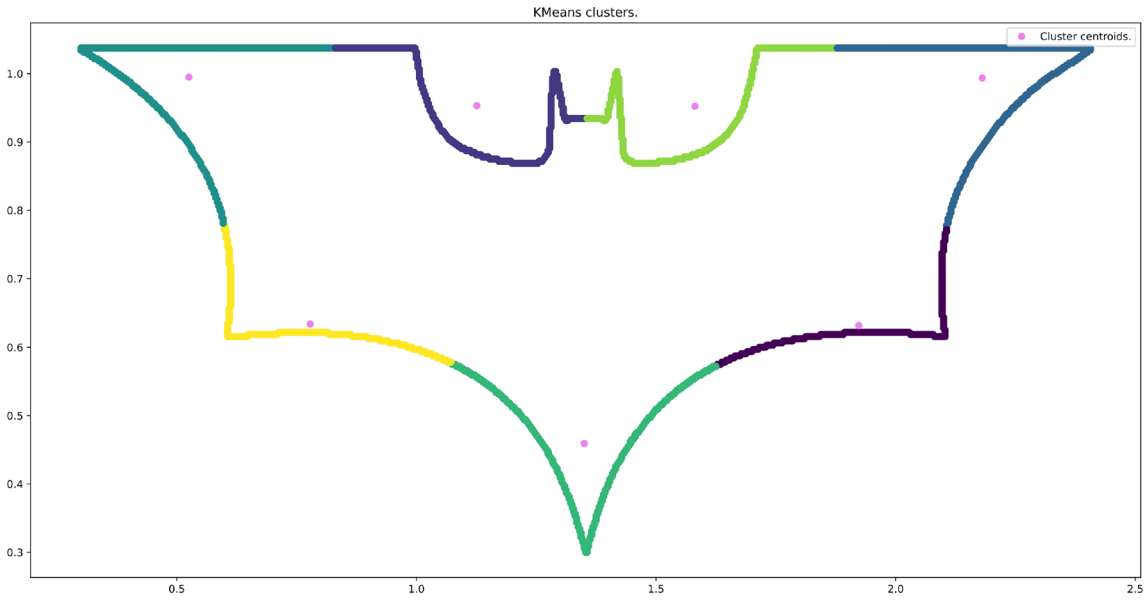

Fig. 12 Clustering results. The waypoints are distributed into a known number of clusters, which is equal to the number of MAVs available. The centroids of the clusters are considered the ideal positions for the MAV swarm to emulate the image

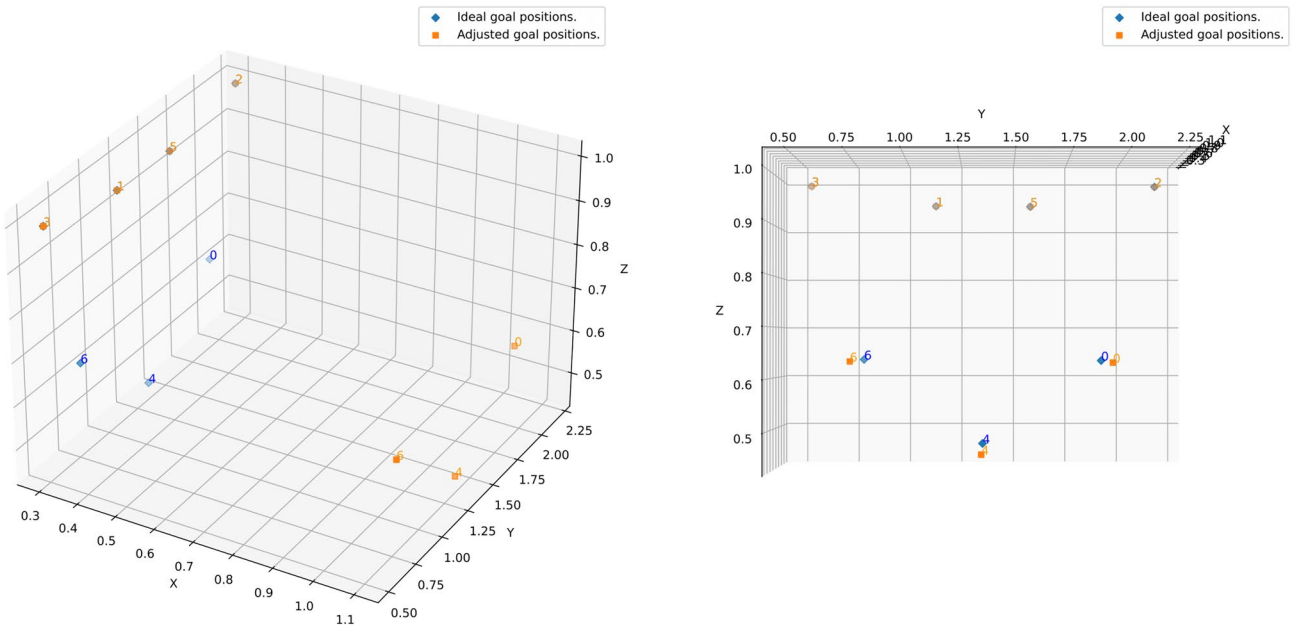

Fig. 13 Results of cost function optimization. The final positions are computed to avoid between-drone collisions and are represented as orange dots. A front view allows us to check whether the displacement from the ideal positions shown as blue dots warps the image reconstruction

Physical Experiment

The results of our physical experiments are shown in Fig. 15 for an MDF of seven MAVs for an input image of a bat silhouette. Since prior work on generating flight paths from digital images is limited, we could not compare the evaluation metrics and the results from this work with those of other works in the literature, as the I/O of these systems differs. We propose the evaluation methodology described

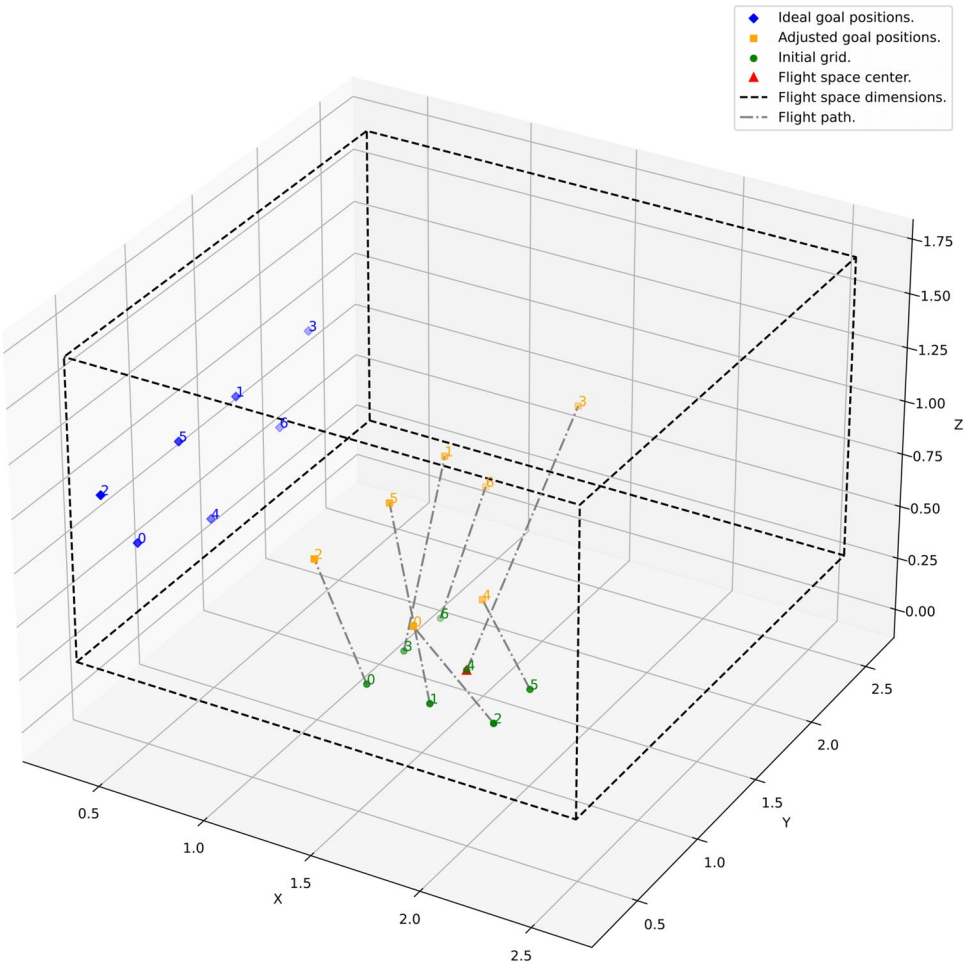

Fig. 14 Calculated MDF from an input image of a bat silhouette. The green circles represent the initial grid of MAVs; the red triangle shows the centre of the flight space; the black dashed lines show the dimensions of the flight space; and the dashed/dotted grey lines show the flight paths of the MAVs. This figure allows us to check the different MAV/position assignments shown with grey lines; for instance, MAV 0 is assigned to position 2

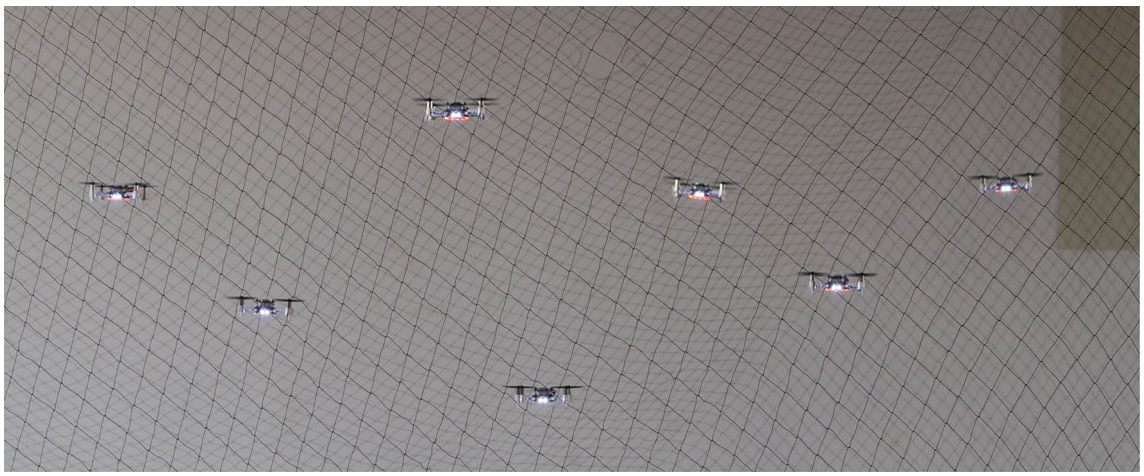

Fig. 15 Real-life results for the MDF system. The system generates a bat silhouette as intended

Table 1 Evaluation metrics for light painting mode in physical drones

Metric	Test images						Average
	A	Batman	Boat 2	Circle	Heart	Holi	
RMSE	0.087034	0.157847	0.284034	0.179877	0.154584	0.417562	0.213490
Mean	0.080214	0.137918	0.242925	0.166769	0.138483	0.355412	0.186954
Median	0.080742	0.135265	0.208714	0.160938	0.137564	0.304038	0.171210
Standard deviation	0.033775	0.076774	0.147182	0.067407	0.068693	0.219182	0.102169
Minimum	0.025966	0.027454	0.024886	0.035550	0.015202	0.013244	0.023717
Maximum	0.182111	0.321177	0.727020	0.317374	0.310204	1.036357	0.482374

Table 2 Evaluation metrics for light painting mode in simulated drones

Metric	Test images						Average
	A	Batman	Boat 2	Circle	Heart	Holi	
RMSE	0.146744	0.128745	0.319023	0.019805	0.019607	0.182799	0.136121
Mean	0.135465	0.112889	0.284876	0.017571	0.018496	0.145958	0.119209
Median	0.123634	0.104618	0.257089	0.015938	0.019302	0.116380	0.106160
Standard deviation	0.056418	0.061896	0.143601	0.009136	0.006504	0.110054	0.064602
Minimum	0.024481	0.009298	0.020198	0.001404	0.003700	0.001245	0.010054
Maximum	0.292981	0.229668	0.641984	0.047737	0.032369	0.408934	0.275612

in the next section as a baseline for comparing future work or further improvements to the scheme set out in this paper.

Evaluation

This section describes the evaluation methods used to measure the performance of CrazyKhoreia in both the LP and MDF modes of use, and present a summary of the obtained metrics. An open-source implementation of our system is provided in a publicly available Github repository.[1]

For validation and testing purposes, we used the Crazyswarm Project's Crazyflies [25]. The Crazyswarm Project includes a ROS [26] server that manages the Crazyflie fleet, and allows control with Python scripting using ROS communication protocols for both high and low-level tasks.

Crazyflies have an onboard state estimation module that uses one-way ultra-wideband communication [27]. This module localizes the MAVs within a flight space 2.89 × 2.89 × 2.0 m in size, with an Ultra-Wideband Sensor (UWB) in each corner and one mounted on each MAV. It also runs an Extended Kalman Filter (EKF) using the UWB sensor array and the Time Difference of Arrival (TDoA V2) mode. The expected accuracy of the sensor, according to the manufacturer Bitcraze, is approximately 0.1 m, although this accuracy is heavily dependent on the flight space distribution, as well as the electromagnetic environment, and this system may have an average error of around 0.2 m.[2]

A Sony α6400 is used to capture long-exposure photographs for the LP mode of use and to record the MDF flights.

Light Painting

To obtain the observed trajectories for the MAVs, we logged the state estimator output at 10 Hz to compare them with CrazyKhoreia's output as ground truth at a level of detail of 0 m. We used Crazyflie's firmware as software-in-the-loop to simulate the dynamics of the MAV, this allowed us to further validate the LP mode with simulation data, sampled at 5 Hz. The datasets were sampled at different frequencies and are not synchronized, meaning that it is not possible to compare them without aligning the data.

Sturm et al. [28] were faced with a similar situation, as they needed to evaluate a large sequence of RGB-D images sampled at 30 Hz against a ground truth trajectory from a motion-capture system sampled at 100 Hz. They proposed an automatic evaluation method for associating and evaluating the two drifted datasets by analyzing the global consistency of the estimated trajectory against the ground truth and calculating the Root Mean Squared Error (RMSE). We use these algorithms[3] to associate the estimator logs with the ground truth to evaluate our system. Table 1 shows a summary of these metrics for six flight trials with diverse

[1] https://github.com/santiagorg2401/crazyKhoreia.

[2] https://www.bitcraze.io/documentation/system/positioning/accuracy-loco/.

[3] https://cvg.cit.tum.de/data/datasets/rgbd-dataset/tools.

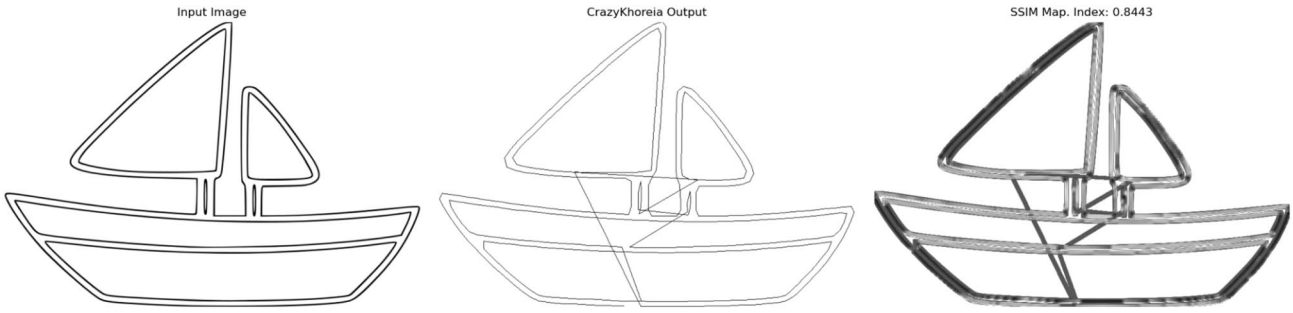

Fig. 16 SSIM on the boat test image. Additional light traces between contours degrade image fidelity, which lowers the SSIM

Table 3 SSIM analysis on all test images

	Test images						
Metric	A	Batman	Boat 2	Circle	Heart	Holi	Average
SSIM Index	0.8674	0.9131	0.8443	0.9653	0.9801	0.8434	0.9023

input images such as a capital letter A, a bat silhouette, a boat, a circle, a heart and the word "holi", using physical drones. Whereas Table 2 shows the results of these trials in simulation mode.

Although metrics such as RMSE and MSE are commonly used for path planning, and we used them in this article to evaluate the flight component of this work, these metrics are not a true measure of how the waypoint generator CrazyKhoreia recreates the input image. For this reason, we use the Structural Similarity Index Measure, or SSIM [29]. The SSIM measures the differences between a reference image and its altered version in a range from 0 to 1 to obtain a perceptual image quality metric. We applied this computation by first creating a zeros image matching the shape of the original image, then taking the output of the CrazyKhoreia system and scaling it back to match the dimensions of the figure, and finally tracing a path in the waypoints coordinates. In Fig. 16, we show how this method works with the boat image, which contains several contours. The SSIM values for all testing images range from 0.9801 to 0.8674, successfully validating the image fidelity of the waypoint generator. Table 3 shows the results for different test images, where images with multiple contours contribute to a lower similarity measure due to undesired light traces.

Additionally, the influence of the detail level can also be measured using the SSIM. As seen in Fig. 9, the resulting image fidelity rapidly degrades as the detail level increases. We tested this by comparing the output of CrazyKhoreia for detail level ranging from 0.00 to 0.20 against the original image, resulting in SSIM values from 0.8744 to 0.7290 as expected. In Table4, we present the SSIM value and number of waypoints for each detail level on the boat figure.

Finally, these metrics show that the flight performance of CrazyKhoreia in LP mode is aligned with the dynamic constraints of the Crazyflies. However, an additional error is introduced by the complexity of the image; this can be observed in the "Holi" and"Boat 2"trials, which had a larger error than the previous tests and a lower SSIM. The MAV also has to cross unexpected space to change between contours, which adds light traces which are not desirable and introduces further error into the evaluation of the system. Furthermore, the flight instability due to the small size of the flight space and aerodynamic issues does not allow the MAV to fly consistently, resulting in an experimental error that is approximately %56.84 larger than the simulated error.

Table 4 SSIM analysis on different detail levels on the boat image

	Detail levels				
Metric	0.00	0.05	0.10	0.15	0.20
SSIM Index	0.8744	0.8024	0.7956	0.7488	0.7290
Number of Waypoints	7597	288	144	110	67

Multi-Drone Formation

In MDF, the MAVs are meant to remain at a set position, and the metrics were measured by taking a picture like Fig. 15. By using Image-J [30], we could estimate the swarm position in the YZ plane and calculate the error using the Euclidean distance between the observed and the expected data; we then evaluated the Mean Absolute Error (MAE), Mean Squared Error (MSE) and the Root Mean Squared Error (RMSE). The evaluation results show an even distribution of errors, as the MAE is larger than the MSE, which contributes to a less warped MAV formation and thus better replicates the original image. However, as the RMSE and the MSE suggest, the system is still sensitive to outlier errors

Table 5 Evaluation metrics for multi-drone formation mode in physical drones

Metric	Test images				Average
	Triangle 3 MAVs	Triangle 4 MAVs	Star 5 MAVs	Batman 7 MAVs	
MAE	0.033	0.100	0.140	0.157	0.108
MSE	0.003	0.015	0.026	0.041	0.021
RMSE	0.058	0.122	0.161	0.204	0.136

that may impact the quality of the MDF mode. The results are summarized in Table 5.

In this case, MDF involves the MAVs hovering indefinitely until the user tells them to land. The control system of the MAV and its stability over time significantly contribute to the overall system error. In this case, CrazyKhoreia's performance in the MDF mode was highly accurate, with an RSME of 0.136 m, which matches the EKF accuracy.

On the other hand, we tested the MDF system in a simulated environment using 5, 15, 25, 50, 75, and 100 MAVs. However, it was challenging to perform simulations from 50 UAVs and higher since the simulator had not been tested with large swarms yet, causing outliers to be far from their goal position or failing to simulate completely. We documented these experiments in Fig. 17, showing a plot containing the goal positions in blue, and the observed positions from the simulation log in orange. Although the impact of increasing the number of MAVs is feasible, the computational time increases considerably with larger swarms, as seen in Fig. 18. We condense the metrics evaluation on simulated experiments in Table 6. Besides higher positional error and computational cost, CrazyKhoreia still achieves feasible results regardless of the number of MAVs.

Discussion

In recent years, there has been a growing interest in developing systems for automating aerial light shows with UAVs. Particularly, the focus of this work was generating motion based on digital images as this offers a straight-forward way to create aerial light shows in a fast manner. In Table 7 we summarize the most relevant work in motion planning based on a user input, such as images, drawing, curves and endpoints. To the best of our knowledge, there has not been a consensus in a standardized way to evaluate this type of systems. Hence, we compare our work based on its features, and the evaluation methods performed. Our work can function with a single or multiple UAVs, and it is evaluated quantitatively by using common metrics such as RMSE, MAE and MSE.

Pichierri [19] in their work, CrazyChoir, offers a similar pipeline to creating coordinated motion using UAVs. We consider their contribution on an LP system as it is most relevant in this context. They created a toolbox capable of tracking cursor movement in a GUI, which they later transformed into waypoints. They evaluated the system by creating a test flight with a CrazyFlie MAV and judging the resulted motion quality visually. Fagundes [18] also offered an approach to LP design by utilizing images and parametric curves as inputs. In this case, they used a laplacian filter to detect the image edges, which they later used to control the UAV motion. As [19], they also analized the resulted LP image visually to evaluate their performance.

Moreover, Weng [20] proposed an interesting way to take advantage of three different viewing angles for the audience by building a 3D ambigram. They managed to do this by taking three input images and the dimensional conditions of the scenario. Then, these images are processed using binarization and a linear conversion to translate it into the visual space. They optimized the resulting waypoints by using a projection error to match common waypoints between the three images and using them to build the MDF. The authors reported a MAE of 0.03765, 0.02246 and 0.02593 for the three test images used in the simulated experiments. Jan [21] tackled computational issues commonly found in aerial light shows. For instance, they created a system which takes already processed waypoints and handles task assignment, coordination, fault tolerance by providing backup drone, and collision avoidance.

Additionally, we performed a semi-quantitative evaluation on the systems provided on [18] and [19]. For this, we used the images provided in their letters for both inputs and outputs. Then, we obtained SSIMs for both cases, which can be compared against the evaluation metrics for this paper. The corresponding SSIM values were 0.5564, 0.4579, and 0.8434 for Pichierri, Fagundes and the"holi"trial for CrazyKhoreia, which was the highest complexity image, respectively. It is worth noting that in both cases, the system is sensitive to flight performance, which affects image fidelity. In Fig. 19, we show the SSIM for both cases, with an index of 0.5564 and 0.4579 for Pichierri and Fagundes, respectively.

Conclusion

In this paper, we have introduced CrazyKhoreia, a functional and user-friendly robotic perception architecture for MAV path planning in light shows based on high-level user input. Our algorithms have been fully implemented in Python

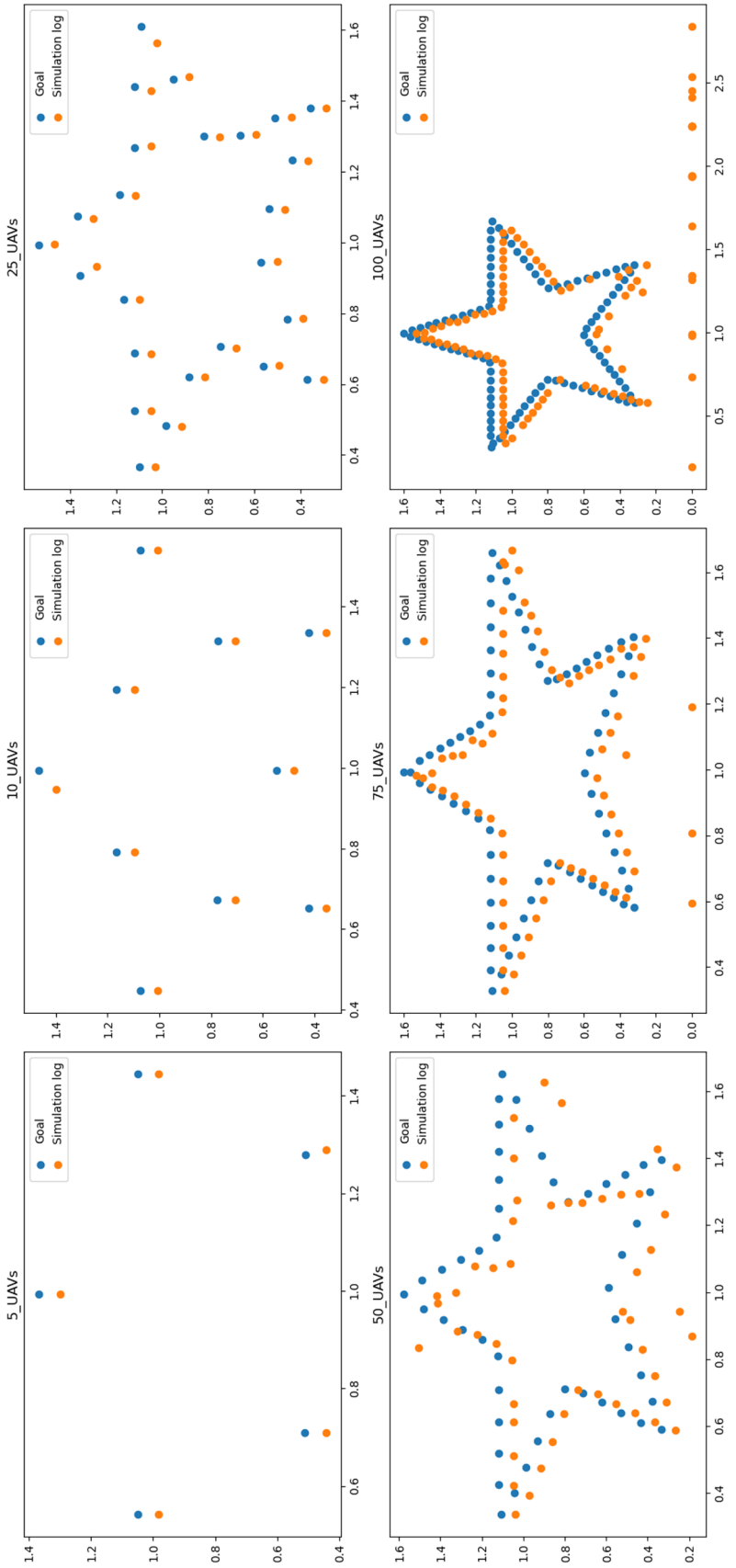

Fig. 17 Performance test varying the number of simulated UAVs from 5 up to 100

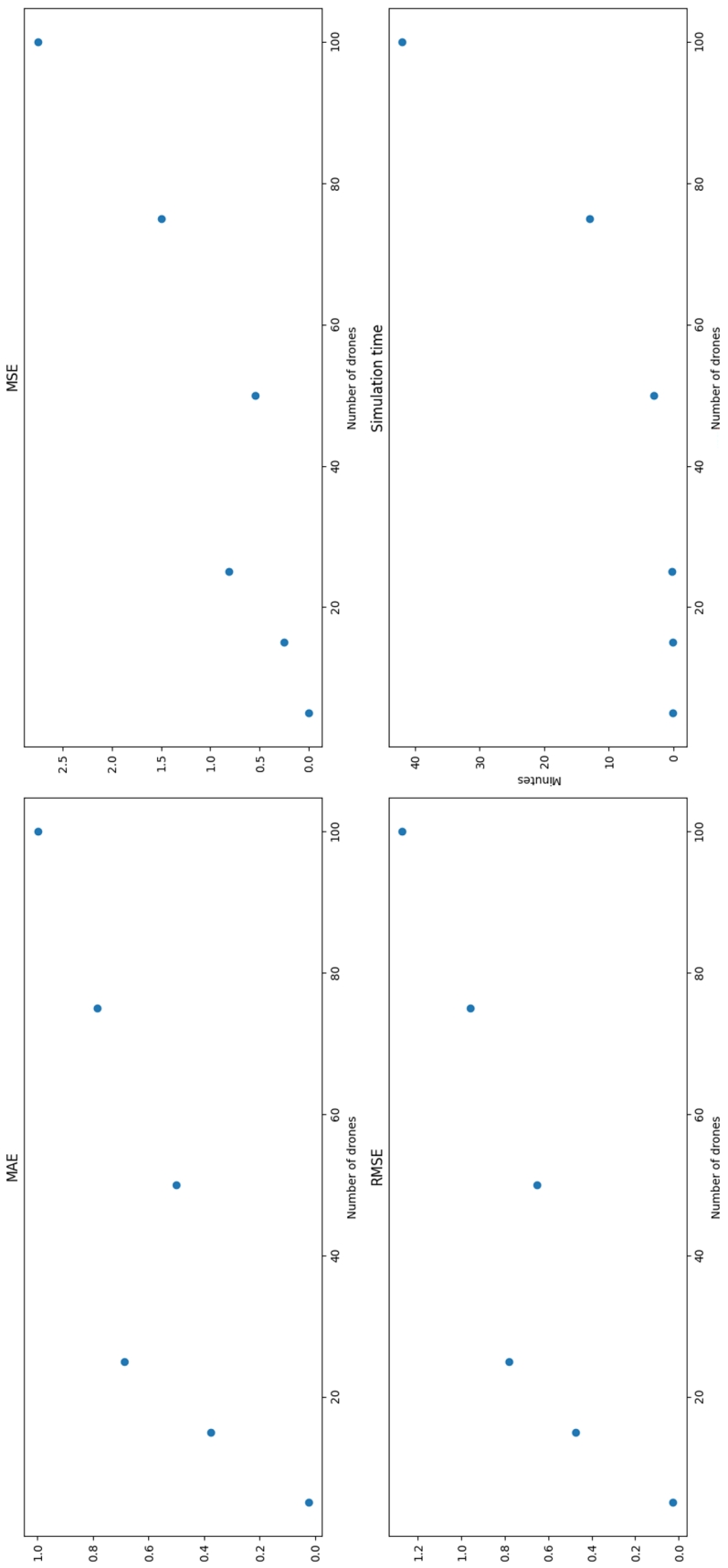

Fig. 18 Plots of the error variation relative to the number of UAVs

Table 6 Evaluation metrics for multi-drone formation mode in simulated drones

	Number of MAVs					
Metric	5	15	25	50	75	100
MAE	0.024146	0.375684	0.685345	0.500909	0.785833	0.998290
MSE	0.001540	0.248626	0.810208	0.541254	1.498744	2.752233
RMSE	0.026157	0.475007	0.779889	0.650511	0.960280	1.271988
Compute time	0.033333	0.083333	0.250000	3.000000	13.00000	42.00000

Table 7 Comparisson between state-of-the-art aerial light shows methods and CrazyKhoreia

		Mode		Evaluation		
	Input type	Single UAV	Multi UAV	Qualitative	Quantitative	Method
CrazyKhoreia	Images	✓	✓	✗	✓	RMSE, MAE, and MSE
Pichierri [19]	Cursor tracking	✓	✗	✓	✗	Visual
Fagundes [18]	Parametric curves and images	✓	✗	✓	✗	Visual
Weng [20]	Images	✗	✓	✗	✓	MAE and MPE
Jan [21]	End-points	✗	✓	✗	✓	Complexety

Fig. 19 SSIM comparison between Pichierri [19] and Fagundes [18] respectively

and are publicly available as an open-sourced PyPi package called CrazyKhoreia, making the code easily accessible for further use and development. This work offers the user a pipeline from inputting a digital image to creating a fully autonomous MAV swarm that executes the trajectories planned by the algorithms proposed in this paper. Our system is capable of designing and processing an aerial MAV light show in two modes of operation: LP, in which a single MAV draws an image using a light bulb, and MDF mode, in which multiple MAVs hover at predetermined positions to accurately resemble the image.

We found that CrazyKhoreia's flight performance in the LP mode was adequate in terms of the swarm's accuracy on the logs of physical experiments such as the letter A and the Batman logo, which had RMSE errors of 0.087034 and 0.157847 m, respectively compared against the output of CrazyKhoreia with a detail level of 0.02. Moreover, SSIM analysis on the waypoints generator shows how CrazyKhoreia can recreate the original image with SSIM values from 0.9801 to 0.8434, with higher values meaning a better structural similarity between the input and resulting images. However, on more complex images

such as Boat2 and Holi, the system gave RMSE errors of around 0.284034 and 0.417562 m. An average RMSE of 0.213490 is found in physical experiments while simulation provided an error of 0.136121 m. This was due to flight instabilities and aerodynamic issues, and because the MAVs flew through areas that were not in the original image, which increased the error. In contrast, in the MDF mode, the system performed above our expectations, with an RMSE error of 0.136 m, which is considerably lower than that of the UWB sensors and the EKF accuracy provided by the manufacturer. Testing the MDF mode using simulated drones proved CrazyKhoreia's scalability. It can successfully run on swarms of 5 to 100 MAVs. However, large swarms affect error and computational time. Also, simulation proved challenging on large swarms, increasing the number of outliers and further affecting evaluation metrics, with RMSE values ranging from 0.026157 to 1.271988 m.

Future work will involve creating an optimization function for the LP mode to minimize the length of the flight path between contours and reduce power consumption. Even though the MDF mode works as intended, it would be interesting to allow the system to automatically change between formations smoothly to enable more vivid aerial light shows. Furthermore, studying different geometrical representations for the collision avoidance algorithm might improve the MDF performance of large swarms in dense environments. Additionally, as this is an offline planner, CrazyKhoreia does not account for external disturbances. As observed in the physical experiments, small environmental or navigational perturbations affect how the image is recreated by the drone. Future work can include accounting for the effects on image fidelity of factors such as wind and localisation drift. Notably, adding an onboard controller capable of reacting to these perturbations while minimizing the error would increase the system's robustness and feasibility for outdoor deployment. At present, CrazyKhoreia operates independently of the controller or hardware used, and it can be integrated with standard controllers such as PID, or more advanced control strategies that explicitly address external disturbances.

Funding No funding was received to assist with the preparation of this manuscript.

Data Availability Data sharing not applicable to this article as no datasets were generated or analyzed during the current study.

Declarations

Conflict of interest The authors have no conflict of interest to declare that are relevant to the content of this article.

References

1. Fennelly LJ, Perry MA. Appendix 16a - unmanned aerial vehicle (drone) usage in the 21st century. In: Davies, S.J., Fennelly, L.J. (eds.) The Professional Protection Officer (Second Edition), Second edition edn., pp. 183–189. Butterworth-Heinemann, Boston. 2020. https://doi.org/10.1016/B978-0-12-817748-8.00050-X . https://www.sciencedirect.com/science/article/pii/B978012817748800050X.
2. Sun H, Qi J, Wu C, Wang M. Path planning for dense drone formation based on modified artificial potential fields. In: 2020 39th Chinese Control Conference (CCC), 2020;pp. 4658–4664. https://doi.org/10.23919/CCC50068.2020.9189345.
3. Premebida C, Ambrus R, Marton Z-C. Intelligent robotic perception systems. In: Hurtado, E.G. (ed.) Applications of Mobile Robots. IntechOpen, Rijeka. 2018. Chap. 6. https://doi.org/10.5772/intechopen.79742.
4. Ruiz MS, Vargas AMP, Cano VR. Detection and tracking of a landing platform for aerial robotics applications. In: 2018 IEEE 2nd Colombian Conference on Robotics and Automation (CCRA), 2018;pp. 1–6. https://doi.org/10.1109/CCRA.2018.8588112.
5. Saavedra-Ruiz M, Pinto-Vargas AM, Romero-Cano V. Monocular visual autonomous landing system for quadcopter drones using software in the loop. IEEE Aerosp Electron Syst Mag. 2022;37(5):2–16. https://doi.org/10.1109/MAES.2021.3115208.
6. Restrepo-García S, Romero-Cano V. CrazyKhoreia, a robotic perception system for UAV path planning from digital images. TechRxiv. 2023. https://doi.org/10.36227/TECHRXIV.21836868.V1.
7. Du X, Luis CE, Vukosavljev M, Schoellig AP. Fast and in sync: Periodic swarm patterns for quadrotors. In: IEEE Int. Conf. Robot. Autom., 2019;pp. 9143–9149. https://doi.org/10.1109/ICRA.2019.8794017.
8. Hsu C-F, Kao W-H, Chen W-Y, Wong K-Y. Motion planning and control of a picture-based drawing robot system. In: IFSA-SCIS 2017 - Jt. 17th World Congr. Int. Fuzzy Syst. Assoc. 9th Int. Conf. Soft Comput. Intell. Syst., 2017;pp. 1–5. https://doi.org/10.1109/IFSA-SCIS.2017.8023232.
9. Canny J. A computational approach to edge detection. IEEE Trans. Pattern Anal. Mach. Intell. 1986;**PAMI-8**(6), 679–698. https://doi.org/10.1109/TPAMI.1986.4767851.
10. Suzuki S, be K. Topological structural analysis of digitized binary images by border following. Comput graph image process. 1985;30(1):32–46. https://doi.org/10.1016/0734-189X(85)90016-7.
11. Stoppel S, Bruckner S. Lineslab: A flexible low-cost approach for the generation of physical monochrome art. Comput Graph Forum. 2019;38(6):110–24. https://doi.org/10.1111/cgf.13609. https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.13609
12. Galea B, Kia E, Aird N, Kry PG. Stippling with aerial robots. In: Proceedings of the Joint Symposium on Computational Aesthetics

and Sketch Based Interfaces and Modeling and Non-Photorealistic Animation and Rendering. Expresive '16, pp. 125–134. Eurographics Association, Goslar, DEU. 2016.
13. Serpiva V, Karmanova E, Fedoseev A, Perminov S, Tsetserukou D. Dronepaint: Swarm light painting with dnn-based gesture recognition. In: ACM SIGGRAPH 2021 Emerging Technologies. SIGGRAPH '21. Association for Computing Machinery, New York, NY, USA. 2021. https://doi.org/10.1145/3450550.3465349.
14. Ibrahimov R, Zherdev N, Tsetserukou D. Dronelight: Drone draws in the air using long exposure light painting and ml. In: 2020 29th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN), 2020;pp. 446–450. https://doi.org/10.1109/RO-MAN47096.2020.9223601.
15. Gebhardt C, Hepp B, Nägeli T, Stevšić S, Hilliges O. Airways: Optimization-based planning of quadrotor trajectories according to high-level user goals. In: Conf. Hum. Factors Comput. Syst. - Proc. CHI '16, pp. 2508–2519. Association for Computing Machinery, New York, NY, USA. 2016. https://doi.org/10.1145/2858036.2858353.
16. Zheng Y, Yang S, Liu X, Wang J, Norton T, Chen J, Tan Y. The computational fluid dynamic modeling of downwash flow field for a six-rotor uav. Front Agric Sci Eng. 2018;5(2):159. https://doi.org/10.15302/J-FASE-2018216.
17. Zhou D, Wang Z, Bandyopadhyay S, Schwager M. Fast, on-line collision avoidance for dynamic vehicles using buffered voronoi cells. IEEE Robot Autom Lett. 2017;2(2):1047–54. https://doi.org/10.1109/LRA.2017.2656241.
18. Fagundes LA, Barcelos CO, Vassallo RF, Brandão AS. Exploring the science and art of uav light painting: From equations and pixels to long-exposure photography. 2024 International Conference on Unmanned Aircraft Systems, ICUAS 2024, 2024;755–762. https://doi.org/10.1109/ICUAS60882.2024.10556858.
19. Pichierri L, Testa A, Notarstefano G. Crazychoir: Flying swarms of crazyflie quadrotors in ros 2. IEEE Robotics and Automation Letters. 2023;8:4713–20. https://doi.org/10.1109/LRA.2023.3286814.
20. Weng KC, Lin ST, Hu CC, Soong RT, Chi MT. Multi-view approach for drone light show. Visual Computer. 2023;39:5797–808. https://doi.org/10.1007/S00371-022-02696-8/FIGURES/18.
21. Jan GE, Lei T, Sun CC, You ZY, Luo C. On the problems of drone formation and light shows. IEEE Trans Consum Electron. 2024. https://doi.org/10.1109/TCE.2024.3421516.
22. Otsu N. A threshold selection method from gray-level histograms. IEEE Trans Syst Man Cybern: Syst. 1979;9(1):62–6. https://doi.org/10.1109/TSMC.1979.4310076.
23. Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V, Vanderplas J, Passos A, Cournapeau D, Brucher M, Perrot M. Duchesnay: Scikit-learn: Machine learning in python. J Mach Learn Res. 2011;12(85):2825–30.
24. Nar D, Kotecha R. Optimal waypoint assignment for designing drone light show formations. Results Control Optim. 2022;9:100174. https://doi.org/10.1016/j.rico.2022.100174.
25. Preiss JA, Honig W, Sukhatme GS, Ayanian N. Crazyswarm: A large nano-quadcopter swarm. In: IEEE Int. Conf. Robot. Autom., 2017;pp. 3299–3304. https://doi.org/10.1109/ICRA.2017.7989376.
26. Quigley M, Conley K, Gerkey B, Faust J, Foote T, Leibs J, Wheeler R, Ng AY, *et al.* Ros: an open-source robot operating system. In: Proc. - IEEE Int. Conf. Robot. Autom., 2009;vol. 3, p. 5. Kobe, Japan.
27. Ledergerber A, Hamer M, D'Andrea R. A robot self-localization system using one-way ultra-wideband communication. In: IEEE Int. Conf. Intell. Robots Syst., 2015;pp. 3131–3137. https://doi.org/10.1109/IROS.2015.7353810.
28. Sturm J, Engelhard N, Endres F, Burgard W, Cremers D. A benchmark for the evaluation of rgb-d slam systems. In: IEEE Int. Conf. Intell. Robots Syst., 2012;pp. 573–580. https://doi.org/10.1109/IROS.2012.6385773.
29. Wang Z, Bovik AC, Sheikh HR, Simoncelli EP. Image quality assessment: from error visibility to structural similarity. IEEE Trans Image Process. 2004;13(4):600–12. https://doi.org/10.1109/TIP.2003.819861.
30. Abràmoff MD, Magalhães PJ, Ram SJ. Image processing with imagej Biophotonics Int. 2004;11(7):36–42.